Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

August 31, 2022

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

BUSD Pay

0x4af1a841f9ac1bc559a12314bf61140f04463b28

Table of Contents

Audit Summary

Project Name	BUSD Pay
Website	https://busdpay.org
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x4af1a841f9ac1bc559a12314bf61140f04463b28
Audit Method	Static Analysis, Manual Review
Date of Audit	31 August 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Source Code

Coinsult was comissioned by BUSD Pay to perform an audit based on the following code:

https://bscscan.com/address/0x4af1a841f9ac1bc559a12314bf61140f04463b28#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

ContractChecker SAFU

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x2c7015a66c8e06b2325f19ada51cf92344cc7a8a	1,000,000,000	100.0000%

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
Informational	Does not compromise the functionality of the contract in any way
Low-Risk	Won't cause any problems, but can be adjusted for improvement
Medium-Risk	Will likely cause problems and it is recommended to adjust
High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
Informational	0	0	0	0
Low-Risk	7	7	0	0
Medium-Risk	0	0	0	0
High-Risk	0	0	0	0

Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	Owner cannot mint new tokens
Owner can blacklist?	Owner cannot blacklist addresses
Owner can set fees > 25%?	Owner cannot set the sell fee to 25% or higher
Owner can exclude from fees?	Owner can exclude from fees
Owner can pause trading?	Owner cannot pause the contract
Owner can set Max TX amount?	Owner can set max transaction amount

More owner priviliges are listed later in the report.

Error Code	Description
SWC-116	CWE-829: Inclusion of Functionality from Untrusted Control Sphere

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
secondsUntilAutoClaimAvailable = nextClaimTime &gt; block.timestamp ?
                                                 nextClaimTime.sub(block.timestamp) :
                                                 0;
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
function updateGasForProcessing(uint256 newValue) public onlyOwner {
    require(newValue &gt;= 200000 &amp;&amp; newValue &lt;= 500000, &quot;gasForProcessing must be betw
    require(newValue != gasForProcessing, &quot;Cannot update gasForProcessing to same value&quot;);
    emit GasForProcessingUpdated(newValue, gasForProcessing);
    gasForProcessing = newValue;
}
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While `1_ether` looks like `1 ether`, it is `10 ether`. As a result, it's likely to be used incorrectly.

Error Code	Description
SLT: 056	Missing Zero Address Validation

Low-Risk: Could be fixed, will not bring problems.

No zero address validation for some functions

Detect missing zero address validation.

```
function changeMarketingWallet(address _marketingWallet) external onlyOwner {
    require(_marketingWallet != marketingWallet, "Marketing wallet is already that address");
    require(!isContract(_marketingWallet), "Marketing wallet cannot be a contract");
    marketingWallet = _marketingWallet;
    emit MarketingWalletChanged(marketingWallet);
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

Error Code	Description
SLT: 016	Functions that send Ether to arbitrary destinations

Low-Risk: Could be fixed, will not bring problems.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function sendBNB(address payable recipient, uint256 amount) internal {
    require(address(this).balance &gt;= amount, "Address: insufficient balance");

    (bool success, ) = recipient.call{value: amount}("");
    require(success, "Address: unable to send value, recipient may have reverted");
}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

Error Code	Description
SWC-104	CWE-252: Unchecked Return Value

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function claimStuckTokens(address token) external onlyOwner {
    require(token != address(this), "Owner cannot claim native tokens");
    if (token == address(0x0)) {
        payable(msg.sender).transfer(address(this).balance);
        return;
    }
    IERC20 ERC20token = IERC20(token);
    uint256 balance = ERC20token.balanceOf(address(this));
    ERC20token.transfer(msg.sender, balance);
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

Error Code	Description
SLT: 038	Imprecise arithmetic operations order

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 fees = (amount * _totalFees) / 1000;
amount = amount - fees;

if (burnTaxShare &gt; 0)
{
    uint256 burnTaxTokens = (fees * burnTaxShare) / _totalFees;
    fees -= burnTaxTokens;
    super._transfer(from,DEAD,burnTaxTokens);
}
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Error Code	Description
SLT: 062	Comparison to boolean constant

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Boolean equality

Detects the comparison to boolean constants.

```
if (maxWalletLimitEnabled)
{
    if (_isExcludedFromMaxWalletLimit[from]  == false &amp;amp;&amp;amp;
        _isExcludedFromMaxWalletLimit[to]    == false &amp;amp;&amp;amp;
        to != uniswapV2Pair
    ) {
        uint balance  = balanceOf(to);
        require(
            balance + amount &amp;lt;= maxWalletAmount,
            &amp;quot;MaxWallet: Recipient exceeds the maxWalletAmount&amp;quot;
        );
    }
}
```

Recommendation

Remove the equality to the boolean constant.

Exploit scenario

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
           // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compare to `true` or `false`.

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	🟢 Owner cannot set the transfer fee to 25% or higher
Buy fee	🟢 Owner cannot set the buy fee to 25% or higher
Sell fee	🟢 Owner cannot set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	-
Max buy fee	-
Max sell fee	-

Contract Pausability Check

Error Code	Description
CEN-02	Centralization: Operator Pausability

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	🟢 Owner cannot pause the contract

Max Transaction Amount Check

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🔴 Owner can set max transaction amount

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	Owner can exclude from fees

Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	🟢 Owner cannot mint new tokens

Ability To Blacklist Check

Error Code	Description
CEN-06	Centralization: Operator Dissalows Wallets

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🟢 Owner cannot blacklist addresses

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

Coinsult lists all important contract methods which the owner can interact with.

⚠️ Owner can set max wallet amount

Notes

Notes by BUSD Pay

No notes provided by the team.

Notes by Coinsult

When changing tax fees, multiply the tax fees by 10. So when entering 3%, enter 30 instead of 3. The contract is setup this way.

✅ Max transaction amount sell cannot be lower than current amount

✅ Max wallet percentage cannot be lower than 2%

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract BUSDPay is ERC20, Ownable {

uint256 public rewardFeeOnBuy       = 40;
uint256 public marketingFeeOnBuy    = 40;
uint256 public liquidityFeeOnBuy    = 10;
uint256 public burnTaxFeeOnBuy      = 10;

uint256 public rewardFeeOnSell      = 80;
uint256 public marketingFeeOnSell   = 50;
uint256 public liquidityFeeOnSell   = 10;
uint256 public burnTaxFeeOnSell     = 10;

uint256 public totalBuyFee          = 100;
uint256 public totalSellFee         = 150;
uint256 public walletToWalletFee    = 0;

address public marketingWallet = 0x3C1619D25837390C55850809fCD9a19875b8Be0B;

address public operator;

IUniswapV2Router02 public uniswapV2Router;
address public  uniswapV2Pair;

address private DEAD = 0x000000000000000000000000000000000000dEaD;


bool    private swapping;
uint256 public swapTokensAtAmount;

mapping (address =&gt; bool) private _isExcludedFromFees;

DividendTracker public dividendTracker;
address public constant rewardToken = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;//BUSD
uint256 public gasForProcessing = 300000;

event ExcludeFromFees(address indexed account, bool isExcluded);
event FeesBuyUpdated(uint256 rewardFeeOnBuy,uint256 marketingFeeOnBuy,uint256 liquidityFeeOnBuy,uint256
```

Certificate of Proof

Not KYC verified by Coinsult

BUSD Pay

Audited by Coinsult.net

Date: 31 August 2022

✔ Advanced Manual Smart Contract Audit

Coinsult

coinsult.net

End of report
Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net